AD-A038 200 MINNESOTA UNIV MINNEAPOLIS MANAGEMENT INFORMATION SY--ETC F/G 9/2
DIFFERENTIAL FILES: THEIR APPLICATION TO THE MAINTENANCE OF LAR--ETC(U)
JAN 76 D G SEVERANCE, G M LOHMAN

UNCLASSIFIED MISRC-WP-76-05 NL

1 OF 1
AD A038200

END
DATE FILMED
4-77

(11) NW

MISRC

Management Information Systems
Research Center

WORKING PAPER SERIES

MISRC/Graduate School of Business Administration/93 Blegen Hall/University of Minnesota/Minneapolis, Minnesota 55455
(612—373-7822)

WORKING PAPER SERIES

(11) Jan 76 (12) 23 p.

(14) MISRC-WP-76-05

(6) Differential Files: Their Application To the Maintenance of Large Databases.

(9) Working paper

(10) Prepared by

Dennis G. Severance
Associate Professor
Graduate School of Business Administration
University of Minnesota

and

Guy M. Lohman
Graduate Student
Department of Operations Research
Cornell University
Ithaca, New York

Management Information Systems Research Center
Graduate School of Business Administration
University of Minnesota
Minneapolis, Minnesota 55455

Abstract

DIFFERENTIAL FILES: THEIR APPLICATION TO THE MAINTENANCE OF LARGE DATABASES

Dennis G. Severance
University of Minnesota

Guy M. Lohman
Cornell University

The representation of a collection of data in terms of its differences from some pre-established point of reference is a basic compaction technique which finds wide applicability. This paper describes a differential database representation which is shown to be an efficient method for storing large and volatile databases. The technique confines database modifications to a relatively small area of physical storage and as a result offers two significant operational advantages. First, because the "reference point" for the database is inherently static, it can be simply and efficiently stored. Moreover, since all modifications to the database are physically localized, the process of backup and the process of recovery are relatively fast and inexpensive.

Key Words and Phrases: Database maintenance, data sharing, backup and recovery, differential files

CR Categories: 3.50, 3.73, 3.80, 4.33

DIFFERENTIAL FILES: THEIR APPLICATION TO THE MAINTENANCE OF LARGE DATABASES

Dennis G. Severance
University of Minnesota

Guy M. Lohman
Cornell University

1. INTRODUCTION

The representation of data in terms of differences from a pre-established point of reference is a data compaction technique with wide applicability. The differential coding of satellite data, the pre-execution merge of an object module with an associated "patch" deck, the invocation of recursive function calls, the modification of a DO-loop index, the concept of base-addressing, and even the distribution of a revised system manual in the form of errata sheets are all applications of a common principle: differential encoding.

This paper describes a differential database representation which is shown to be an efficient method for storing a large and changing database. The power of the representation is derived from the fact that all database modifications are localized into a relatively small storage area, called a differential file. By consolidating changes in this manner, it is possible to reduce backup costs, speed the process of database recovery, and even minimize the probability of a serious data loss. In addition, the technique can provide increased data availability while reducing both storage and retrieval costs.

The paper consists of two major sections: Section 2 motivates, describes, and analyzes the concept of a differential file; Section 3 presents ten specific advantages of the idea. A summary and an outline of future research are given in conclusion.

2. DIFFERENTIAL FILES

2.1 An analogy. A differential file for a database is analogous to an errata list for a book. Rather than print a new edition of a book each time a change in text is desired, a publisher will identify corrections by page and line number, and collect them into an errata list which is distributed with each book. This procedure significantly reduces publication costs. To reference the corrected version of the book, however, readers must consult the errata list before any reading of the main text. An increase in access time is thus traded for a decrease in maintenance cost. If text changes are continued, the errata list will grow to a sufficient length that reorganization costs are justified. All changes would then be incorporated into the book, forming a new physical edition.

Updating a large database poses a similar problem. As with a book, it is generally simplest and least expensive to accumulate changes over a period of time and to post them en masse when creating a new database edition (generation). It is most expensive, as measured in terms of storage costs, maintenance time, and overall system complexity, to directly modify the database with each update transaction. As a compromise, a differential file can be used like an errata list to collect and identify pending record changes. Consulting the differential file as a first step in data retrieval effectively yields an up-to-date database. At a cost of increased access time, system overhead may be reduced. When the differential file grows sufficiently large, a reorganization would incorporate all changes into a new generation of the database, and the now empty differential file would begin accumulating changes anew.

2.2 Earlier proposals. The concept of a differential file has been rediscovered many times. The authors have benefited from numerous discussions with

colleagues who have seen the idea used in one form or another to solve particular updating problems. Three documented systems will be described. Turnburke [16] outlines a differential structure for tape systems which is designed to avoid the writing of unchanged data records while sequentially processing batched updates. A data file is composed from two ordered subfiles: a large collection of read-only records is stored on one tape, while a smaller collection of modified records is maintained on a separate "change-tape". To update the data file, both tapes are merged with a transaction file and a new change-tape is output. Unchanged records from the read-only tape are never written. Turnburke recommends data file organization once one half of all records have been modified.

Roycroft [13] suggests a direct access file organization which also makes use of a differential file concept to process file changes. The system addresses records via a unique identifier, and every data reference passes through a database index which points to all records. Once created, the main data file is never modified. New database records are accessed through the index but are stored in a separate overflow area. All record modifications are treated as record additions. A new copy of the record is created and the index is updated to point into the overflow area. The old record is not destroyed, but rather maintained as a before-image and pointed to by the new record. Roycroft's primary motivation for this technique is to permit a data record to grow in size as a result of an update without disturbing the positioning of neighboring records.

A system with a similar structure is described by Rappaport [12]. It was developed to facilitate database recovery after an electrical power failure. Again all database records are accessed through a system index, and all modifications are physically separated into a file of changes called a MODFILE. Each changed record points back to its before-image. In the event of a power loss, information

in a transaction log is used in conjunction with the MODFILE to undo partially completed update transactions.

2.3 A Generalization. Whenever a record is updated in either the Roycroft or Rappaport system, a record search mechanism (initially associated with only the main data file) is modified to address a new record copy which is stored in what is essentially a different file. As depicted by Figure 1a, the current version of any identified record, whether in the main file or the differential file, is accessed via a common search mechanism -- the system index.

A generalization of this record accessing strategy is shown in Figure 1b. Here, given the identifier for any database record, the differential file is always searched first for that record; in the event that the record is not found, it is then retrieved from the main data file. Implicit in this diagram is the fact that each file may utilize a separate search mechanism. The main file index is then static and can be quickly recovered from a backup copy in the event of a loss. Index volatility is shifted to a smaller, and therefore more quickly recoverable, differential file index.

To isolate the main file and its search mechanism from change, a delay in the form of a differential file search is paid for every record retrieval. If the two data files and their search mechanisms can be assigned to different devices and accessed via separate channels, then both file searches may proceed in parallel and system users will not perceive the increased delay. When such overlap is impossible, one can expect the average time of a data record retrieval (assuming a judicious selection of the differential file search strategy [15]) to increase by the amount of time required for a random access to secondary memory. This additional access time may be comparatively large and can seriously degrade system performance.

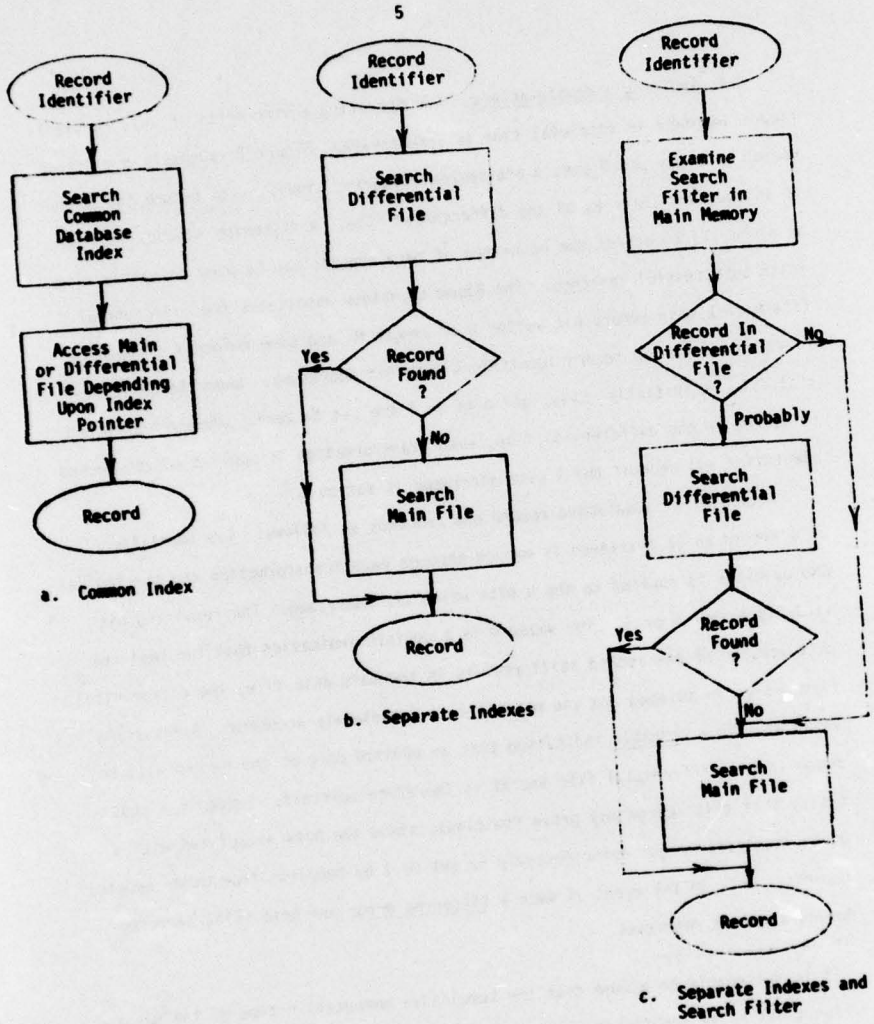

Figure 1 Alternative Access Strategies

2.4 Avoiding a double-access. For operating environments in which a significant increase in retrieval time is intollerable, Figure 1c suggests a modified search strategy which uses a pre-search filtering algorithm to reduce the number of unnecessary searches of the differential file. A filtering scheme, devised by Bloom [1] to detect the occurence of rare events, can be used to nearly eliminate unsuccessful searches. The Bloom technique associates the differential file with a main memory bit vector B of length M, and some number X of hashing functions which map record identifiers into bit addresses. When the differential file is initially empty, all bits in B are set to zero. Whenever a record is stored in the differential file, each transformation is applied to the record identifier and each of the X bits addressed is set to 1.†

Retrieval of a database record now proceeds as follows. The identifier of a record to be retrieved is mapped through each transformation and the logical AND operator is applied to the X bits which are addressed. The resulting bit value is either 0 or 1. The value 0 is a certain indication that the most recent version of the record still resides in the main data file; the differential file search is skipped and the main file is immediately accessed. A resulting value of 1 is a probable indication that an updated copy of the record will be found in the differential file and it is therefore searched. There is a possibility that this search may prove fruitless, since the bits associated with a given identifier might coincidentally be set to 1 by mappings from other updated records. Only in the event of such a filtering error are both files searched during a record retrieval.

---

† It is reasonable to assume that the cumulative computation time of the hashing functions is insignificant in comparison to the time required for an access to secondary memory. Either division [3] or quadratic hashing functions [11], for example, can generate several addresses quickly.

2.5 *Double-access frequency*. The probability of a filtering error at a given point in time is a function of both the proportion of main file records which have been modified and the proportion of bits in B which are set to 1. Expected values for all of these quantities can be calculated. Consider a database of N records. Assume updates are independent, uniformly distributed over all records, and arrive over time at a fixed rate r. Similarly assume the existence of a collection of X hashing functions whose mappings are independent and uniform over B. Define T to be the length of time between reorganizations of the main data file. The expected proportion of distinct main file records, R_t, which are updated during a time period of length t is given by

$$R_t = 1 - \left(\frac{N - 1}{N}\right)^{rt} \approx 1 - \exp\left(\frac{-rt}{N}\right)$$

For various values of record update intensity, rT/N, Figure 2 depicts the growth over time of both R_t and rt/N. Respectively, these curves characterize the size of a differential file which contains (a) only the most recent image of a changed record, or (b) an accumulation of all such record images.

Now consider a random bit in B. The probability that it has value 1 at time t is

$$1 - \left(\frac{M - 1}{M}\right)^{rtX}$$

and therefore, given an identifier of an <u>unchanged</u> record, the probability of a filtering error is

$$\left[1 - \left(\frac{M - 1}{M}\right)^{rtX}\right]^X = \left[1 - \exp\left(\frac{-rtX}{M}\right)\right]^X .$$

The unconditional probability of a filtering error at time t is thus given by

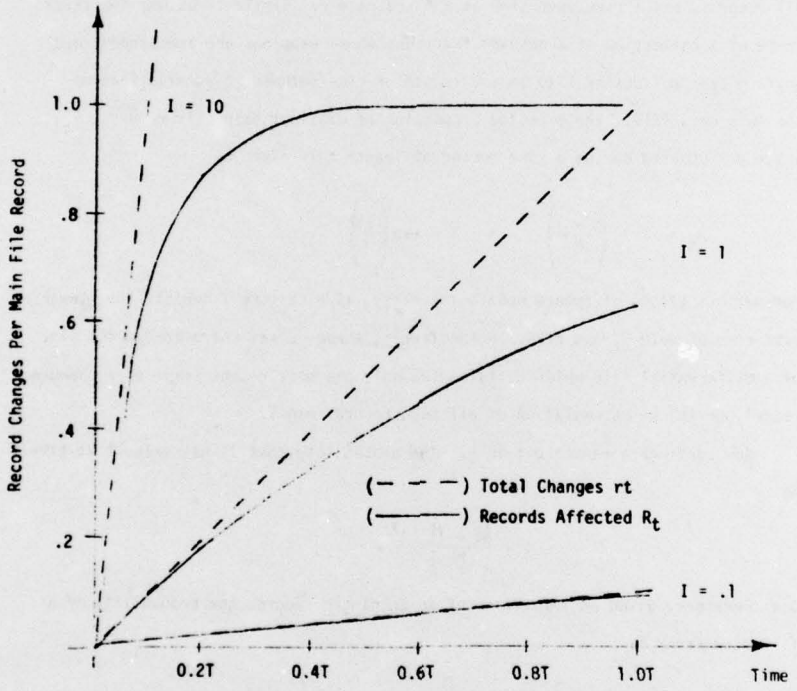

Figure 2 Differential File Growth at
Update Intensity I = rT/N

$$P_t = \exp\left(\frac{-rt}{N}\right)\left[1 - \exp\left(\frac{-rtX}{M}\right)\right]^X$$

2.6 Designing a Bloom filter. The frequency with which filtering errors occur can be controlled by manipulating the values selected for M and X. A common design problem is likely to be the following. Some quantity of space M' is available in main memory and can reasonably be allocated as a bit vector. It is necessary to determine the number of transformations which should map into this fixed area. If the value selected for X is too small, the system will underutilize the available bits. If, on the other hand, X is too large then most bits will be set to 1 and the filter will be ineffective. Figure 3 illustrates P_t as a function of time for various values of X. N and M' are arbitrarily set to rT. Each curve shows an initial increase in filtering errors, which eventually decline as the proportion of unchanged records in the main file diminishes. As the value of X increases, superior initial performance deteriorates at a faster rate to inferior final performance.

In general, for a fixed M', a number of reasonable objective functions might be used to select a value for X. Among these are:

(a) minimize P_t at a given time t',

(b) minimize $\frac{1}{T}\int_0^T P_t\,dt$,

(c) minimize (maximum P_t) over the time interval [0,T],

(d) minimize $\frac{1}{T}\int_0^T P_t\,dt$, subject to $P_t \leq p'$.

Intuitively, one can see from Figure 3 that each of these objectives might lead to the selection of a different value for X. While the formal analysis required by problems (b), (c), and (d) is beyond the scope of the current paper, problem (a) is easily solved using classical optimization techniques. Setting

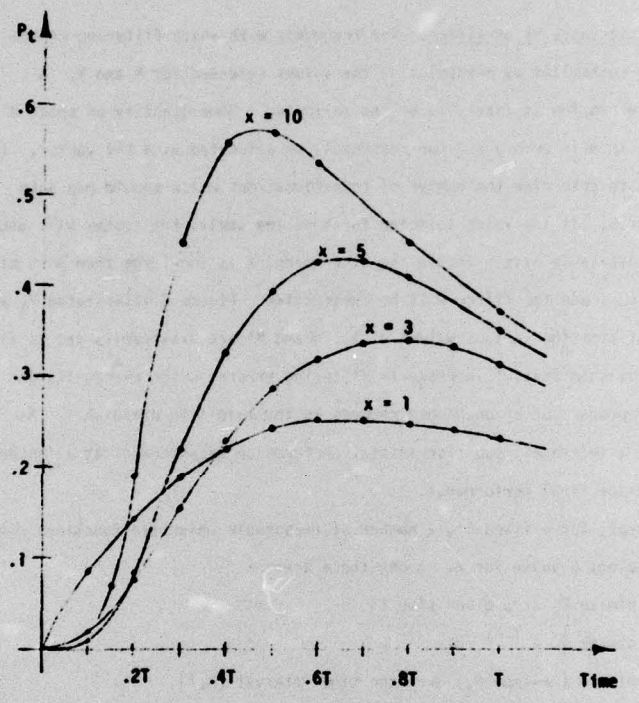

Figure 3 Probability of a Filtering Error with X Transformations. Given M = N = rT.

$$\frac{dP_t}{dX} = \exp\left(\frac{-rt}{N}\right)\left[X\left[1 - \exp\frac{(-rt'X)}{M}\right]^{X-1}\frac{rt'}{M}\exp\frac{(-rt'X)}{M} + \ln\left[1 - \exp\frac{(-rt'X)}{M}\right]\left[1 - \exp\frac{(-rt'X)}{M}\right]^{X}\right].$$

equal to zero, yields the equality

$$\hat{X} = \frac{M'\ln 2}{rt'}.$$

Since $\frac{d^2P_{t'}}{dX^2}$ is positive at $\hat{X}$, this value of X minimizes $P_{t'}$ for given values M', r, and t'.

For X equal to $\hat{X}$, one can easily verify that the expected number of bits in B set to 1 at time t' is M'/2 (in agreement with a fundamental theorem of information theory and a similar result obtained by Bloom), and that the probability of a filtering error at this time is

$$\hat{P}_{t'} = \exp\left(\frac{-rt'}{N}\right)\left(\frac{1}{2}\right)^{\frac{M'\ln 2}{rt'}} = \exp\left(\frac{-rt'}{N}\right)(.6185)^{\frac{M'}{rt'}}$$

In practice X must be integral and the two integers nearest $\hat{X}$ should be checked for minimal $P_{t'}$. It can be shown that one of them must be the optimal value of X.

Applying these results to a specific problem with $N = 10^7$, $r = 10^3$, T = 5, $M = 2.5 * 10^4$, and t' = T, one calculates $\hat{X} = 3.47$ and $\hat{P}_T = .0905$. For X = 3 and 4, P_T takes on values .0918 and .0919 respectively. Thus three transformations used in conjunction with the given 3125 byte Bloom-vector is expected to produce fewer than one-in-ten filtering errors at the time of reorganization. (The average error rate over time is approximately one-in-thirty.) A differential file, therefore, will not appreciably affect average retrieval time for this database.

3. ADVANTAGES OF A DIFFERENTIAL FILE

The history of database development efforts shows design simplicity to be a dominant characteristic of successful implementations. And although the notion of a differential file is conceptually rather simple, in practice, any additional system complexity must be justified by tangible benefits. The potential advantages of a differentially organized database are not widely appreciated; even existing systems are rather narrowly motivated. This section therefore collects and discusses ten general benefits that can be realized. Six relate to database integrity and show that a differential file can reduce backup costs, speed recovery and even minimize the chances of a serious data loss. The final four advantages are operational; a differential file can provide increased data availability and simultaneously reduce storage and retrieval costs. In total these benefits constitute a strong argument for a much wider use of differential files, especially *for the maintenance of very large databases.*

3.1 Reduces database dumping costs. In general, to recover a database which has been physically damaged, some form of roll-forward procedure (Yourdon [17]) is employed: The status of the database, saved at a previous point in time is first reloaded; the cumulative effect of all update transactions processed since that time is then re-established via some abbreviated form of reprocessing. The frequency with which the database is copied to its backup file is a critical parameter in designing such a procedure (Chandy, et al., [4], Drake and Smith [6]). Frequent dumping permits fast recovery, but is associated with a high system overhead.

Since the time required for a dump is proportional to the volume of data copied [8], a differential file can drastically reduce the cost to backup a large database, particularly when the proportion of records changed during a

backup period is small. Consider, for example, a database with 10^7, 500-character records stored in track size blocks on an IBM 3330 disk facility. Suppose updates are applied five days per week, 10 hours per day at a rate of 100 changes per hour. Using a fast dump/restore utility [8] a full database dump would require over six hours to accomplish. On the other hand, even after a full week of processing, a differential file, its bit vector, and a reasonable search mechanism could be dumped in less than two minutes. In total, they would occupy less than 300 tracks of storage as compared to 51 disk packs.

3.2 Facilitates incremental dumping. It is sometimes impractical to dump an entire database at one time. An incremental dumping strategy (Sayani [14], also called "differential disk dumping" by Yourdon [17]) will sequence through physical sections of a database, periodically dumping each section which has changed. A differential file implementation in which new records are sequentially allocated in secondary memory (for example, Rappaport's system [12]) lends itself naturally to such a strategy. To provide a complete database backup at any point in time, one need only append to the current backup file those differential file records created since the last dump. With each incremental dump, one might also choose to save the current status of the differential file bit vector and search index. Alternatively, both could be recovered with a single scan of the restored differential file.

3.3 Permits both realtime dumping and reorganization with concurrent updates. Since a dump represents the instantaneous status of a database at a fixed point in time, conventional backup procedures will prohibit all changes while this snapshot is developed. By dumping only a small differential file, the time during which update transactions are prohibited can be substantially reduced. More importantly, one can avoid completely the need to inhibit change by building a

"differential-differential" file to store record updates which are generated during the differential file dumping process. For most applications, this file will be quite small and reasonably held in main memory. Acting as a "cache" store during the dump, it would be scanned before every retrieval. When the dump is complete, its records would be incorporated into the main differential file. Clearly, the same basic idea will permit online reorganization. Since the generation of a new main file might require a significant amount of time, the differential-differential file would be maintained in secondary memory. It replaces the old differential file when reorganization is complete.

These procedures for dumping and reorganizing a database are particularly appropriate for applications such as airline reservation systems, which require 24-hour, online availability, but which experience periods of reduced traffic intensity. Without locking out updates, either procedure could be activated during a slack period and would act to level the system load.

3.4 Speeds recovery from a "soft" data loss. Damage to storage hardware is not the only cause of data loss. A user program may incorrectly modify a database, or a program error, a system deadlock or a machine failure may abort the processing of an update transaction in the midst of a multi-step procedure (such as, a transfer of funds between bank accounts). The content and/or structural integrity of the database may be damaged by either type of error. Rappaport's Vehical and Driver Information System [12] provides a working example of a differential file (maintained in the form of an online after-image log) which permits rapid system recovery through the rollback of incorrectly processed or partially completed transactions.

3.5 Speeds recovery from a hard data loss. As described above, the use of a differential file can dramatically reduce the cost of dumping a large database. An inexpensive dump procedure can be invoked frequently, which will in turn reduce the average number of changes to be reapplied in the event of a database loss.

3.6 Reduces the risk of a serious data loss. When recovering a database, a properly tuned dump-restore utility can reload a physical dump at nearly the maximum transfer rate of available hardware (on the order of 10^5 characters/second). The major portion of recovery time is then spent individually reapplying updates to a small fraction of the restored records. This small subset of changed records constitutes an "Achiles' heel." Traditional update-in-place file organizations distribute changed records widely over secondary memory; this practice guarantees that even localized physical damage (e.g., a track loss or head crash on a single device of a very large database) will require a lengthy recovery procedure. By concentrating updates in a small physical area, a differential file offers three potential advantages:

(a) The critical exposure area of a database is minimized. Most physical damage can be quickly repaired with a localized backup copy procedure.

(b) The critical area may be allocated to a more reliable device type than is practical for the larger main file.

(c) The small critical area may be duplexed to provide the most valuable redundancy for a marginal increase in operating costs.

3.7 Supports "memo files" efficiently. Accurate online updating of a database requires complex software to provide multiuser access control and to assure data recoverability (King and Collmeyer [9]). To avoid the substantial overhead associated with such software, many "online" systems will actually batch updates for

end-of-day processing. Inventory control systems, for example, can generally tolerate some loss of accuracy during the batching cycle provided data integrity is re-established with each batch run. In systems where a predictable information lag might be exploited (e.g., banking or stock quotation) the memo file concept of Davis [5] can be used to maintain "probably-accurate" data without the need for complex software. The idea is to permit software which does not defend against improbable events (such as concurrent update, system failure, head crash) to update a "scratch pad" copy of the database. At end-of-day the copy is discarded and the updates are reapplied to the "real" database. The use for a differential file here is obvious.

3.8 Simplifies software development. Since the main data file and its associated index are unaffected by updates in a differential file system, this afforts a natural environment for the development and testing of new data processing software. Using two differential files, one can imagine a developmental system and a production system running in parallel with both accessing the same main file but modifying their own differential files. To debug new software, online comparisons could be made between the data values maintained by both systems. For very large databases, where it is either (1) infeasible to create a duplicate copy of the database for experimentation, or (2) it is at least impossible for both copies to be online simultaneously, this use of differential files is particularly important.

3.9 Simplifies main file software. Because the main file is static between reorganizations, the structures required for its storage are inherently simple and efficient. Neither free space nor record linkages are allocated to accomodate record growth, and a greater density of data storage can be achieved when the database is initially loaded [13]. Since the main file is read-only, multiple

access requests may be handled concurrently without requiring the use of a complex protocol to avoid deadlock or errors due to simultaneous write access (Brinch Hansen [2]). Thus if a user program requires access to data which is either (1) known to be constant or (2) relatively stable and absolute currency is non-critical, then such requests may bypass the differential file and safely access only the main file without queueing for private access.

3.10 Reduces future database storage costs. Within the next decade, trillion bit random access mass storage devices will be provided by at least one of several competing technologies. The cost for a dynamic read-write capability is expected to be an order of magnitude higher than the cost of a read-only memory.† The application of the differential file concept in such an operating environment is obvious: the large main data file is read-only. The cost reduction which differential files provide will greatly enlarge the realm of feasible computer-based information systems.

4. SUMMARY AND FURTHER RESEARCH

A differential file is an efficient representation of database updates. By consolidating modifications, and physically isolating them from the main, read-only data file, it is possible to reduce backup costs, speed database recovery, avoid serious data losses, increase data availability, decrease storage costs and speed retrieval operations. The paper provides a number of arguments which should motivate a wider use of differential files, particularly for the maintenance of very large databases.

---

† Based upon presentations made in May 1975 by the Panel on Future Architecture at the Very Large Data Base Conference, Framingham, MA.

The design of a differential file system involves tradeoffs among the costs of update, retrieval, storage, backup, and recovery. Implementation questions currently under investigation by the authors include:

(1) What data should be stored in a differential file? Should it contain complete data records, as suggested here, or only data fields which have changed. Should old versions of a differential file record be overwritten or retained as a before-image?

(2) How frequently should differential file backup and reorganization occur?

(3) How does differential file size and filter error probability grow with the non-random arrival of non-uniform updates? How should the filter parameters M and X be selected?

(4) Given a hierarchy of storage devices, at what levels should the main file, differential file, search mechanisms, and bit vector be stored?

(5) How can differential files be used to facilitate maintenance of distributed databases?

Formal statements of these problems are extremely complex and their solution is difficult. When solved, however, the results may have significant practical value.

ACKNOWLEDGEMENTS

The authors wisn to express their appreciation to Eric Clemons, Thomas Dimock, Gordon Everest, and William Maxwell for numerous helpful discussions during the preparation of this paper. The David W. Taylor Naval Ship Research and Development Center provided support for this research under Contract N00014-75-C-1119.

REFERENCES

1. Bloom, B. H., "Space/Time Trade-offs in Hash Coding with Allowable Errors," Communications of the ACM, 13:7 (July, 1970), 422-426.

2. Brinch Hansen, P., Operating System Principles, Englewood Cliffs, NJ: Prentice-Hall (1973), 55-131.

3. Buchholz, W., "File Organization and Addressing," IBM Systems Journal (June, 1963), 80-111.

4. Chandy, K. M., J. C. Browne, C. W. Dissly, and W. R. Uhrig, "Analytic Models for Rollback and Recovery Strategies in Data Base Systems," IEEE Transactions Software Engineering, SE-1:1 (March, 1975), 100-110.

5. Davis, G. B., Management Information Systems: Conceptual Foundations, Structure and Development, New York: McGraw-Hill Book Company (1974), 278.

6. Drake, R. W. and J. L. Smith, "Some Techniques for File Recovery," Australian Comp. J., 3:4 (November, 1971), 162-170.

7. IBM Corporation, Introduction to IBM Direct-Access Storage Devices and Organization Methods GC20-1649-8, White Plains, NY: IBM Corporation (February, 1974), 20-33.

8. Innovation Data Processing Incorporated, Fast Dump Restore and Data Set Functions, User Documentation, Clifton, NJ: Innovation Data Processing Inc. (July, 1973).

9. King, P. F. and A. J. Collmeyer, "Database Sharing -- An Efficient Mechanism for Supporting Concurrent Processes," Proceedings of the 1973 AFIPS Conference (1973), 271-275.

10. Knuth, D. E., Sorting and Searching, The Art of Computer Programming 3, Reading, MA: Addison-Wesley (1973), 561-562.

11. Maurer, W. D., "An Improved Hash Code for Scatter Storage," Communications of the ACM, 11:1 (January, 1968), 35-38.

12. Rappaport, R. L., "File Structure Design to Facilitate On-line Instantaneous Updating," Proceedings of the 1975 ACM SIGMOD Conference, 1-14.

13. Roycroft, A. J., "Techniques for Handling Variable Length Logical Records on IBM Direct Access Storage Devices," Proceedings FILE68 International Seminar on File Organization, Copenhagen (1968), 701-720.

14. Sayani, H. H., "Restart and Recovery in Transaction-Oriented Information Processing System," Proceedings of the 1974 ACM SIGMOD Workshop on Data Description, Access, and Control (May, 1974), 351-366.

15. Severance, D. G., and R. A. Duhne, "Practitioner's Guide to Addressing Algorithms," Communications of the ACM (March, 1976).

16. Turnburke, V. P., Jr., "Sequential Data Processing Design," IBM Systems Journal (March, 1963), 37-48.

17. Yourdon, E., Design of On-Line Computer Systems, Englewood Cliffs, NJ: Prentice-Hall (1972), 340-353, 515-542.